D. Beyrak, student
V. Chuhov, PhD in Engr., As. Prof., research advisor
S. Kobzar, Senior Lecturer, language advisor
Zhytomyr State Technological University

DYNAMIC PLOT OF DIGITIZED DATA IN WOLFRAM MATHEMATICA 10

In spite of the fact that nowadays most of the modern physical quantity measuring devices are digital, for this kind of concerns analog devices are also used. Consequently, there raises the need for digitizing, saving and further use of the captured data. This can be done if a device has the analog output.

In this paper we will focus on the problem of convenient processing of the digitized data array drawing on the example of computation program Wolfram Mathematica 10, considering the typical steps of transforming an original set of values $\{a_1, a_2, \ldots\}$ into the graphic rendition. The issues of choosing and setting up the analog-digital converter are not covered here.

Besides having the actual set of values, it is also necessary to know the end values for every axis to be able to make a plot with correct scales. These values can be measured directly with the analog device and then taken into account in the program during the plotting.

Suppose the digitized signal is obtained in *.wav* format. Then to import it as a set of points in variable **`data`**, it is possible to use the **`Import`** function with specificator **`"Data"`**. For example:

```
data = Import["C:\\Directory\\file.wav", "Data"]
```

As a result, there have to be the set of the kind shown in Figure 1.

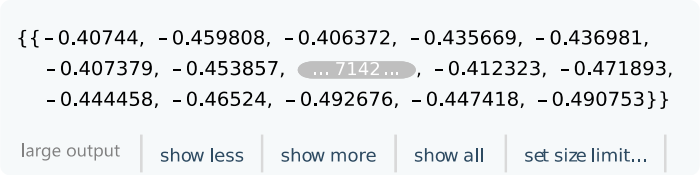

Figure 1. Imported set of values

Note that the original set of data can be stored in other formats. Visit the help page of the **`Import`** function to see the rest of the ways of importing by placing the mouse cursor on the function and pressing the “F1” key.

Sometimes it may be handy to normalize the set of original values in range from 0 to 1. This can be done by using the **`Rescale`** function. Besides that, the measurements are often made in logarithmic scale, so it is worth making delogarithmation of original set. These actions can be done with the following functions:

```
ndata = Rescale[data];
delogdata = 20^ndata;
```

Next, let us generate the set of points, which suits the scale of the x-axis. We will store the lowest and the highest axis points in variables **`xmin`** and **`xmax`** accordingly and

use the **Range** function for obtaining the set. The **Length** function is used here for getting the cardinality value of the original set, and the equation, it appears in, defines the generating step:

```
xmin = 25.86; xmax = 37.5;
axisx = Range[xmin, xmax, (xmax - xmin)/Length[data[[1]]]];
```

Now let us construct the y-axis. To do this we will define the limits as in the previous example, but use the **Rescale** function for remapping the elements of the original, normalized or delogarithmized set pursuant to the real axis limits. The **MinMax** function is used here for finding the maximum and minimum values of the transmuted set. These values stand as arguments of the **Rescale** function.

```
ymin = 0.3; ymax = 1.125;
axisy = Rescale[delogdata, MinMax[delogdata], {ymin, ymax}][[1]];
```

At this point, the values for each axis are ready to be plotted. But to be able to plot using functions such as **ListPlot**, **ListLogPlot** and alike it is necessary to transform data to the form like $\{\{x_i, y_i\}\}$. It can be done via functions **Riffle** and **Partition**:

```
plotdata = Partition[Riffle[axisx, axisy], 2];
```

Considering that in this example the original data are in logarithmic scale of the y-axis, let us plot using the **ListLogPlot** function. It can have many arguments, but the most useful that we will utilize are as follows: **ImageSize** for changing the size of the image, **Joined** for joining the plot points, **GridLines** for activating the gridlines and **AxesLabel** for labeling the axes:

```
ListLogPlot[plotdata, ImageSize -> 500, Joined -> True, GridLines -> Automatic, AxesLabel -> {"f, GHz", "A, dB"}]
```

It is important to underscore the fact that to view the output, the semicolon "**;**" at the end of the line has to be absent. Also, sometimes when the amount of points is low there arises the need to "smooth" the plot, so that it does not look broken. This can be done by interpolation, using **InterpolationOrder** as an argument. For the most part its values are in range from 2 to 4. However, if the plot consists of a large amount of points, interpolation is not reasonable.

While taking the plot reading, it can be useful to get rid of the noise, which is an integral part of every measurement. Since usually the useful signal is modulated by high frequency noise, it is possible to do it away with low pass filter (LP filter) function. In addition, note that it is not necessary to feed data into the variable in order to plot. The next example shows how to plot the filtered data without using variables for them.

```
ListLogPlot[Partition[Riffle[axisx, LowpassFilter[axisy, 0.02]], 2], ImageSize -> 500, Joined -> True, GridLines -> Automatic, AxesLabel -> {"f, GHz", "A, dB"}]
```

In this example, the 0.02 filter value was chosen. Apart from the LP filter, Wolfram Mathematica has other functions for "smoothing" plots, which may appear

more suitable for other cases. For instance, **GaussianFilter** and **MeanFilter** can be used. As for the example above the values 3 and 50 respectively should be recommended. The plots of the signal with the noise unfiltered and filtered by the LP filter are shown in Figure 2 a) and b) correspondingly.

a) b)

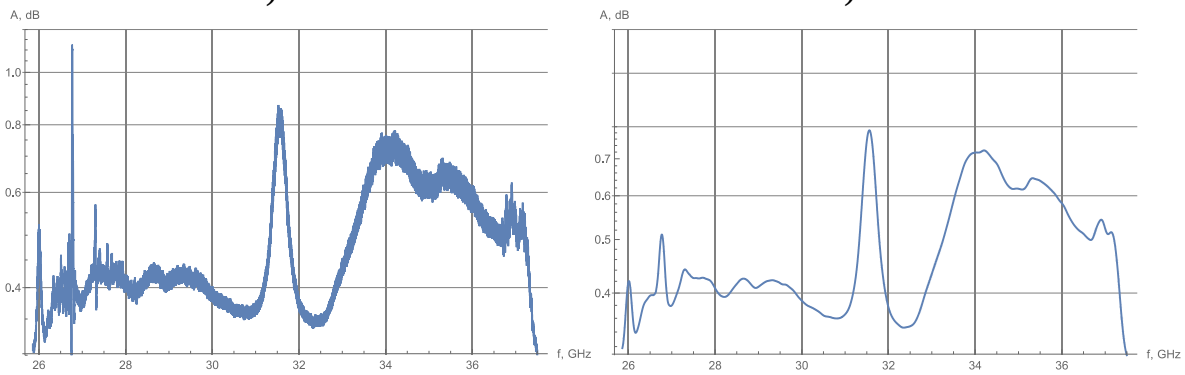

Figure 2. Obtained plots: a) no filtering, b) low-pass filtering

It should be emphasized, when the noise component is significant or causes signal bursts that are not useful, it makes sense to apply LP filter (or its analog) directly to the imported data:

```
data = LowpassFilter[Import["C:\\Directory\\file.wav", "Data"], 0.02]
```

Such method prevents the raise of errors while setting up axes scales.

When it is necessary to compare two or more plots, they can be put on the same coordinate grid. There are a few ways to do that, but we will consider the simplest one — by **Show** function. It can be done in two ways: by putting the corresponding commands into the arbitrary variables beforehand, or straightforwardly. The first method is more elegant, but let us consider both (the notation for **ListLogPlot** is shortened).

```
a = ListLogPlot[plotdata1];
b = ListLogPlot[plotdata2];
Show[{a, b}]
```

or

```
Show[{ListLogPlot[plotdata1],ListLogPlot[plotdata2]}]
```

While taking the readings from the plot it is convenient to use the ability of getting coordinates using mouse cursor. It can be done by right-clicking on the desired plot and choosing **Get Coordinates** menu entry. After that, the cruciate cursor and numerical values of axes will appear.

Another practical capability of dealing with plots is dynamic scaling. It can be implemented with **Manipulate** function. Without getting into details — they can be found in help documentation — let us consider the simplest example of the code, which gives us the two-axes scaling:

```
Manipulate[ListLogPlot[Partition[Riffle[axisx,axisy], 2], ImageSize -> 500, Joined -> True, GridLines -> Automatic, Frame -> False, AxesLabel -> {"f, GHz", "A, dB"}, PlotRange -> {{minx, maxx}, {miny, maxy}}], {{minx,
```

```
xmin, "x-axis min"}, xmin, Mean[{xmin, xmax}]}, {{maxx, xmax, "x-axis max"}, Mean[{xmin, xmax}], xmax}, {{miny, ymin, "y-axis min"}, ymin, Mean[{ymin, ymax}]}, {{maxy, ymax, "y-axis max"}, Mean[{ymin, ymax}], ymax}]
```

The same functionality can also be carried out for combined plots. In order to do this, it is enough to put the argument **PlotRange** into the **ListPlot** function, e.g.:

```
Manipulate[Show[{ListLogPlot[plotdata1, (…), PlotRange -> {{minx, maxx}, {miny, maxy}}], ListLogPlot[plotdata1, (…)], {{minx, xmin, "x-axis min"}, (…)}]
```

In case of multiple measurements with the same device and also for convenience's sake, it is reasonable to take functional approach and roll up all considered operations in one function and use it for dynamic plotting afterwards. It provides the ability to use one short line-length command. However, this method faces strong system requirements. But this problem may be solved by using automatic parallelization function **Parallelize**. Let us consider it for our example:

```
Parallelize[quickplot[data_, xmin_, xmax_, ymin_, ymax_] := Manipulate[ListLogPlot[Partition[Riffle[Range[xmin, xmax, (xmax - xmin)/Length[data[[1]]]],Rescale[20^Rescale[data],MinMax[20^Rescale [data]], {ymin, ymax}][[1]]], 2], ImageSize -> 500, Joined -> True, GridLines -> Automatic, Frame -> False, AxesLabel -> {"f, GHz", "A, dB"}, PlotRange -> {{minx, maxx}, {miny, maxy}}], {{minx, xmin, "x-axis min"}, xmin, Mean[{xmin, xmax}]}, {{maxx, xmax, "x-axis max"}, Mean[{xmin, xmax}], xmax}, {{miny, ymin, "y-axis min"}, ymin, Mean[{ymin, ymax}]}, {{maxy, ymax, "y-axis max"}, Mean[{ymin, ymax}], ymax}]]
```

Here **quickplot** is a chosen name of the roll-up function (can be arbitrary), and its arguments **data_**, **xmin_**, **xmax_**, **ymin_**, **ymax_** are the original data set and axes limits accordingly. Let the original data set be in the new variable **newdata**. Then, the plot can be obtained as follows:

```
quickplot[newdata, 25.86, 37.5, 0.3, 1.125]
```

The result is shown in Figure 3. Also, the part of the plot was selected by slider controls.

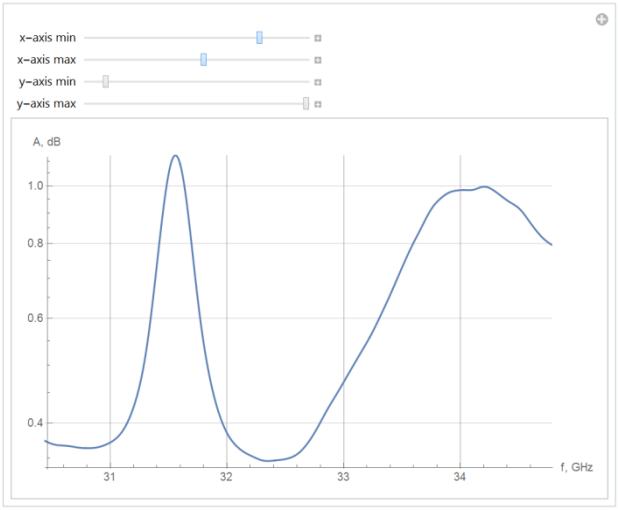

Figure 3. Resulted plot with dynamic scaling

To sum up, the convenient and functional way of digitized data processing was obtained. Moreover, it can be easily modified or corrected if necessary. It is worthy of note that such kind of flexible implementation is provided particularly by great capabilities of the Wolfram Mathematica computation program, which was used in the examples above.